

Herança

SSC0103 - POO

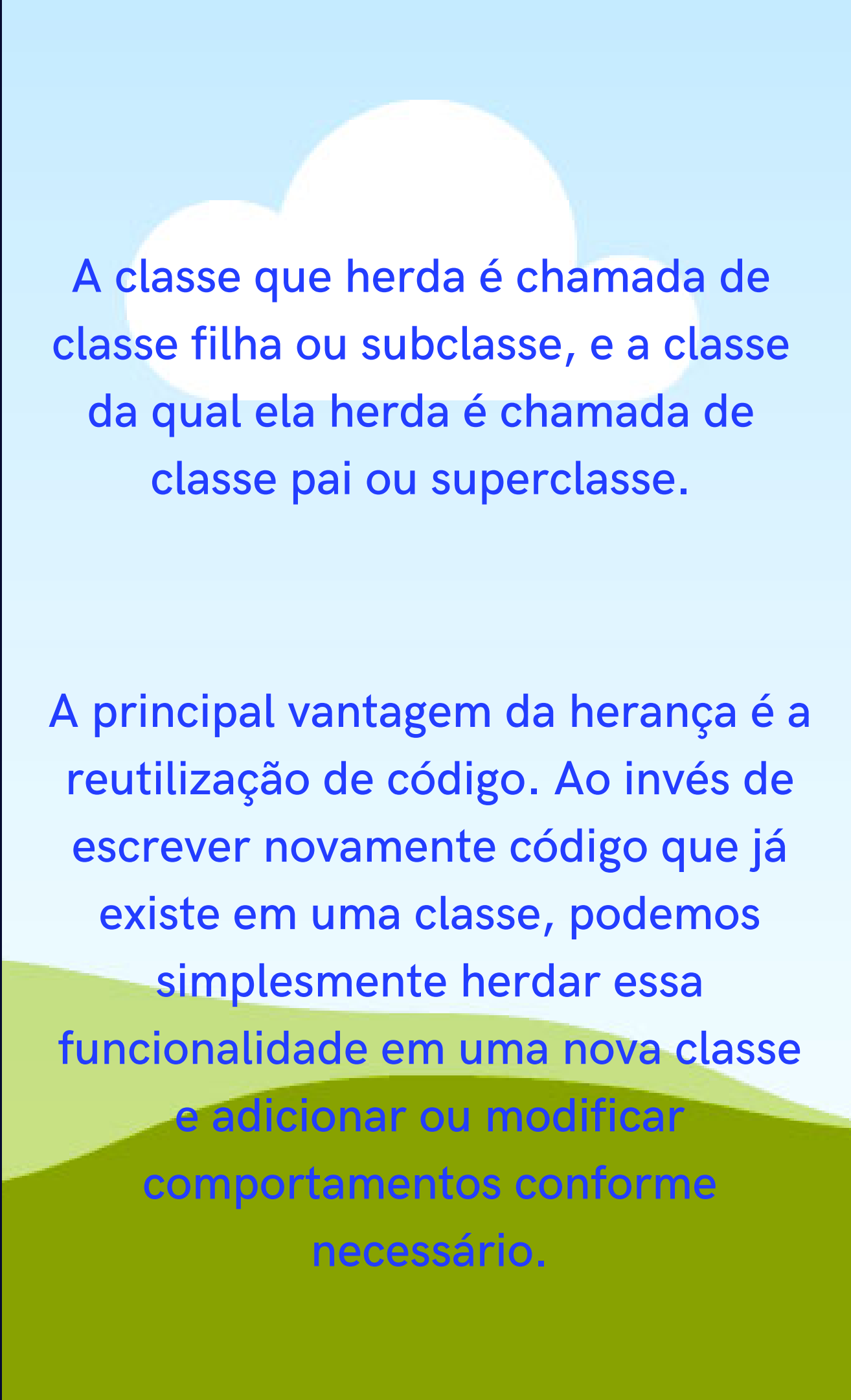
Delamaro

In object-oriented programming, inheritance is when an object or class is based on another object or class. It is a mechanism for code reuse and to allow independent extensions of the original software via public classes and interfaces. The relationships of objects or classes through inheritance give rise to a hierarchy. Inheritance was invented in 1967 for Simula.

Conceito

Herança é um conceito fundamental que permite que uma classe herde atributos e métodos de outra classe.

A herança permite que você crie hierarquias de classes, onde classes mais específicas (subclasses) podem herdar características de classes mais gerais (superclasses), e ao mesmo tempo adicionar novos comportamentos ou substituir comportamentos existentes, se necessário.



A classe que herda é chamada de classe filha ou subclasse, e a classe da qual ela herda é chamada de classe pai ou superclasse.

A principal vantagem da herança é a reutilização de código. Ao invés de escrever novamente código que já existe em uma classe, podemos simplesmente herdar essa funcionalidade em uma nova classe e adicionar ou modificar comportamentos conforme necessário.

Superclasse

```
public class Animal {  
    private String nome;  
  
    public Animal(String nome) {  
        this.nome = nome;  
    }  
  
    public void fazerSom() {  
        System.out.println("O animal faz um som indefinido");  
    }  
  
    public String getNome() {  
        return nome;  
    }  
}
```

Superclasse

```
public class Animal {  
    private String nome;  
  
    public Animal(String nome) {  
        this.nome = nome;  
    }  
  
    public void fazerSom() {  
        System.out.println("O animal faz um som indefinido");  
    }  
  
    public String getNome() {  
        return nome;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Animal animal = new Animal("Bob");  
  
        animal.fazerSom(); // Saída: O animal faz um som indefinido  
        System.out.println(animal.getNome()); // Saída: Bob  
    }  
}
```

Subclasses

```
public class Gato extends Animal {  
  
    public Gato(String nome) {  
        super(nome);  
    }  
  
    @Override  
    public void fazerSom() {  
        System.out.println("O gato mia");  
    }  
  
}
```

```
public class Cachorro extends Animal {  
    public Cachorro(String nome) {  
        super(nome);  
    }  
  
    @Override  
    public void fazerSom() {  
        System.out.println("O cachorro late");  
    }  
  
    public void levantarPata() {  
        System.out.println("Fiz xixi");  
    }  
  
}
```

Subclasses

```
public class Gato extends Animal {  
  
    public Gato(String nome) {  
        super(nome);  
    }  
  
    @Override  
    public void fazerSom() {  
        System.out.println("O gato mia");  
    }  
}
```

```
public class Cachorro extends Animal {  
    public Cachorro(String nome) {  
        super(nome);  
    }  
  
    @Override  
    public void fazerSom() {  
        System.out.println("O cachorro late");  
    }  
  
    public void levantarPata() {  
        System.out.println("Fiz xixi");  
    }  
}
```

Significa que os membros da classe animal também estão na subclasse

Subclasses

```
public class Gato extends Animal {  
  
    public Gato(String nome) {  
        super(nome);  
    }  
  
    @Override  
    public void fazerSom() {  
        System.out.println("O gato mia");  
    }  
  
}
```

```
public class Cachorro extends Animal {  
    public Cachorro(String nome) {  
        super(nome);  
    }  
  
    @Override  
    public void fazerSom() {  
        System.out.println("O cachorro late");  
    }  
  
    public void levantarPata() {  
        System.out.println("Fiz xixi");  
    }  
  
}
```

O método substitui aquele que havia sido definido na superclass

Subclasses


```
public class Cachorro extends Animal {  
    public Cachorro(String nome) {  
        super(nome);  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Cachorro cachorro = new Cachorro("Rex");  
        Gato gato = new Gato("Whiskers");  
  
        cachorro.fazerSom(); // Saída: 0 cachorro late  
        gato.fazerSom();    // Saída: 0 gato mia  
        System.out.println(cachorro.getNome()); // Saída: Rex  
        System.out.println(gato.getNome());    // Saída: Whiskers  
    }  
}
```

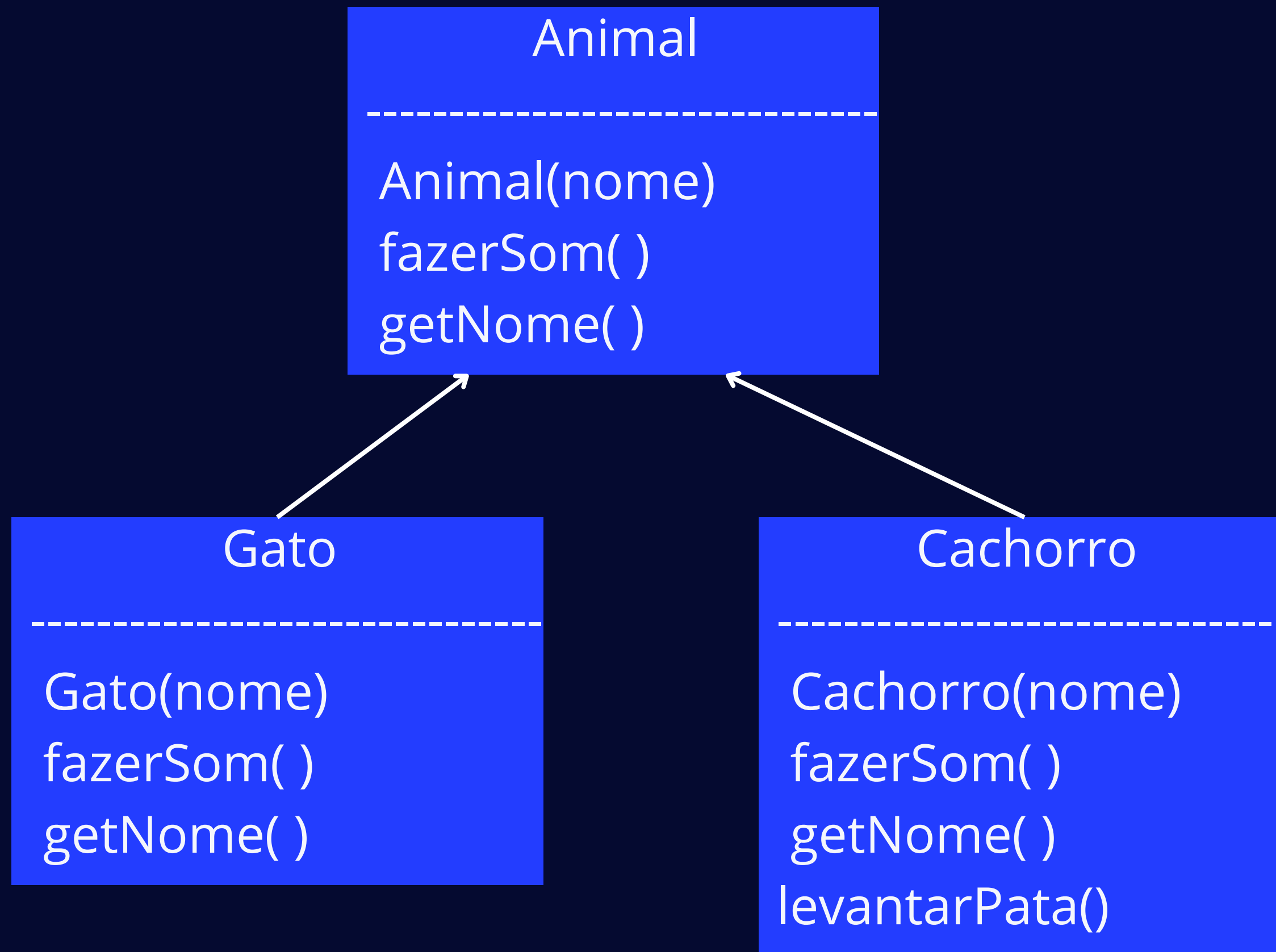
Subclasses

```
public class Cachorro extends Animal {  
    public Cachorro(String nome) {  
        super(nome);  
    }  
}
```

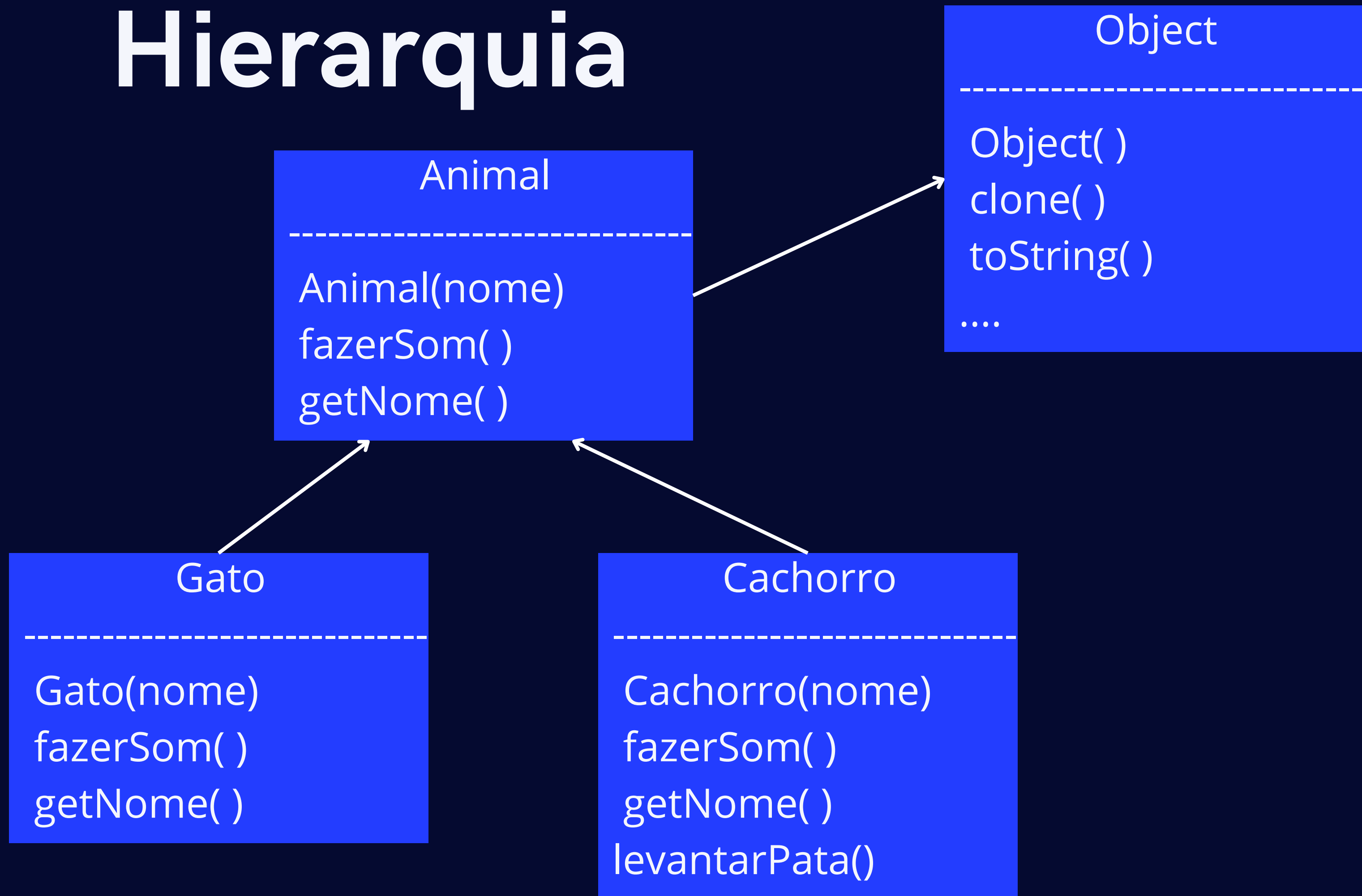
```
public class Main {  
    public static void main(String[] args) {  
        Cachorro cachorro = new Cachorro("Rex");  
        Gato gato = new Gato("Whiskers");  
  
        cachorro.fazerSom(); // Saída: 0 cachorro late  
        gato.fazerSom(); // Saída: 0 gato mia  
        System.out.println(cachorro.getNome()); // Saída: Rex  
        System.out.println(gato.getNome()); // Saída: Whiskers  
        cachorro.levantarPata(); // Saída: Fiz xixi  
    }  
}
```



Hierarquia



Hierarquia



Subclasses

```
public class Gato extends Animal {  
    private String sobrenome;  
  
    public Gato(String nome, String sobre) {  
        super(nome);  
        this.sobrenome = sobre;  
    }  
  
    @Override  
    public void fazerSom() {  
        System.out.println("O gato mia");  
    }  
  
    @Override  
    public String getNome() {  
        return super.getNome() + " " + sobrenome;  
    }  
}
```

```
public class Cachorro extends Animal {  
    public Cachorro(String nome) {  
        super(nome);  
    }  
  
    @Override  
    public void fazerSom() {  
        System.out.println("O cachorro late");  
    }  
  
    public void levantarPata() {  
        System.out.println("Fiz xixi");  
    }  
}
```

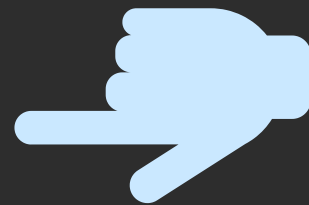
super é usado para chamar o construtor pai e para acessar o método da superclasse.

Subclasses

```
public class Gato extends Animal {  
    private String sobrenome;  
}
```

```
public class Cachorro extends Animal {  
    public Cachorro(String nome) {  
        super(nome);  
    }  
  
    @Override  
    public void fazerSom() {  
        // ...  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Cachorro cachorro = new Cachorro("Rex");  
        Gato gato = new Gato("Whiskers", "de Souza");  
  
        cachorro.fazerSom(); // Saída: O cachorro late  
        gato.fazerSom();     // Saída: O gato mia  
        System.out.println(cachorro.getNome()); // Saída: Rex  
        System.out.println(gato.getNome());     // Saída: Whiskers de Souza  
        cachorro.levantarPata(); // Saída: Fiz xixi  
    }  
}
```



Ninguém adota um animal

```
public abstract class Animal {  
    private String nome;  
  
    public Animal(String nome) {  
        this.nome = nome;  
    }  
  
    public void fazerSom() {  
        System.out.println("0 animal faz um som indefinido");  
    }  
  
    public String getNome() {  
        return nome;  
    }  
}
```


Ninguém adota um animal

```
public abstract class Animal {  
    private String nome;  
  
    public Animal(String nome) {  
        this.nome = nome;  
    }  
  
    public void fazerSom() {  
        System.out.println("O animal faz um som indefinido");  
    }  
  
    public String getNome()  
        return nome;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Animal animal = new Animal("Bob");  
  
        animal.fazerSom(); // Saída: O animal faz um som indefinido  
        System.out.println(animal.getNome()); // Saída: Bob  
    }  
}
```

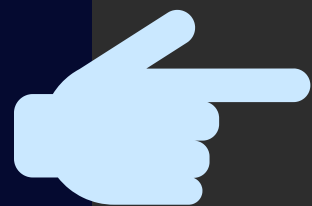
Ninguém adota um animal

```
public abstract class Animal {  
    private String nome;  
  
    public Animal(String nome) {  
        this.nome = nome;  
    }  
  
    public void fazerSom() {  
        System.out.println("O animal faz um som indefinido");  
    }  
  
    public String getNome() {  
        return nome;  
    }  
}
```

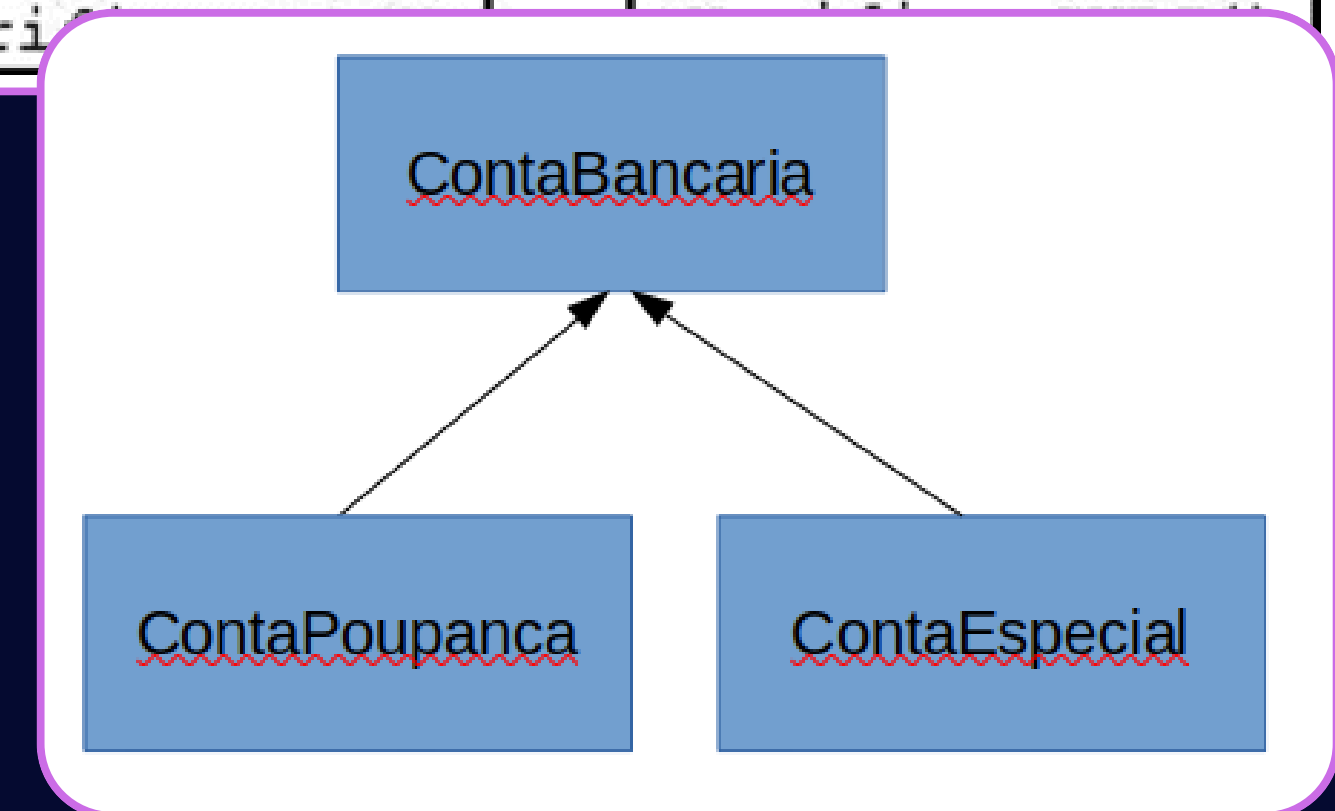
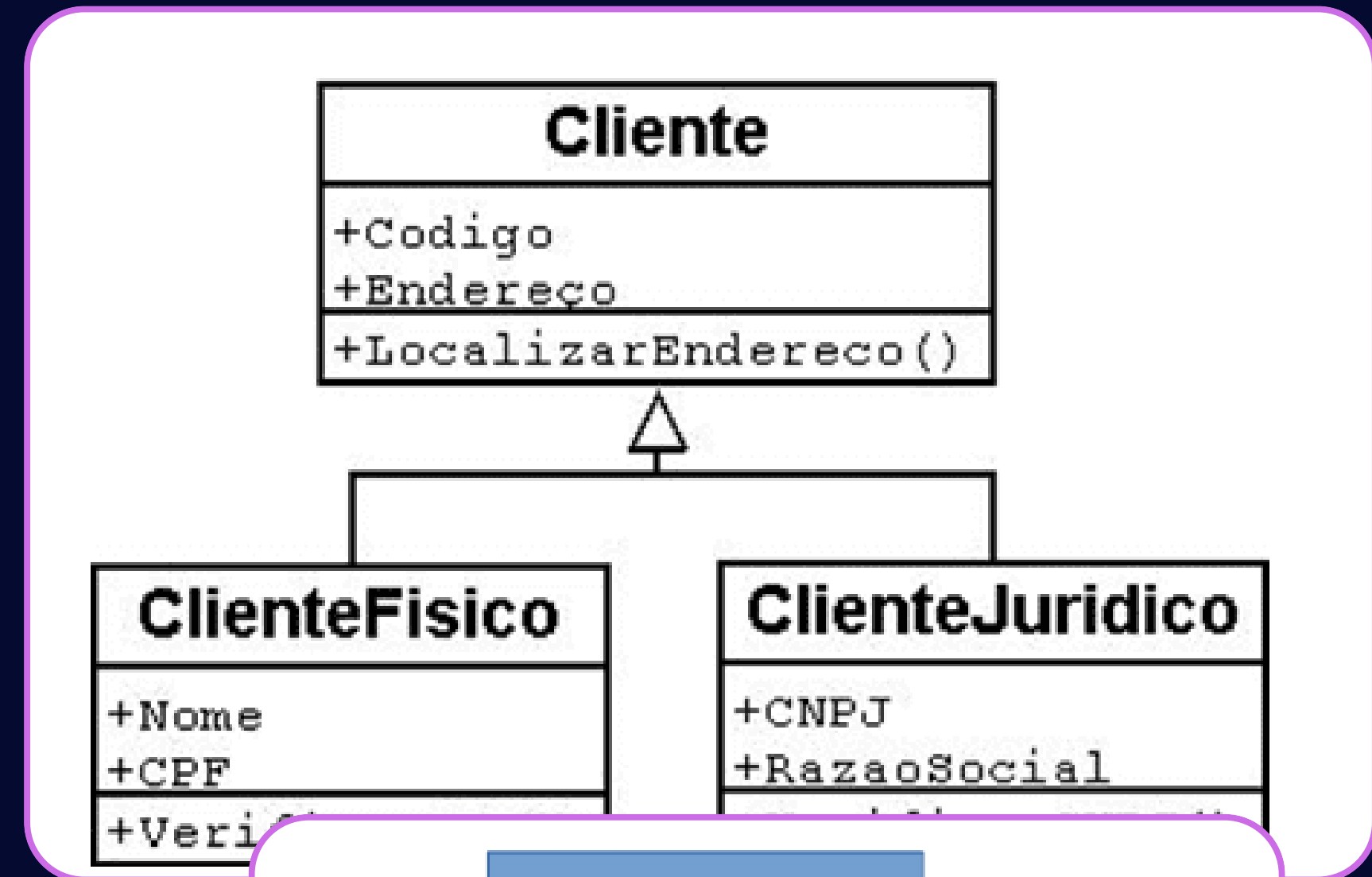
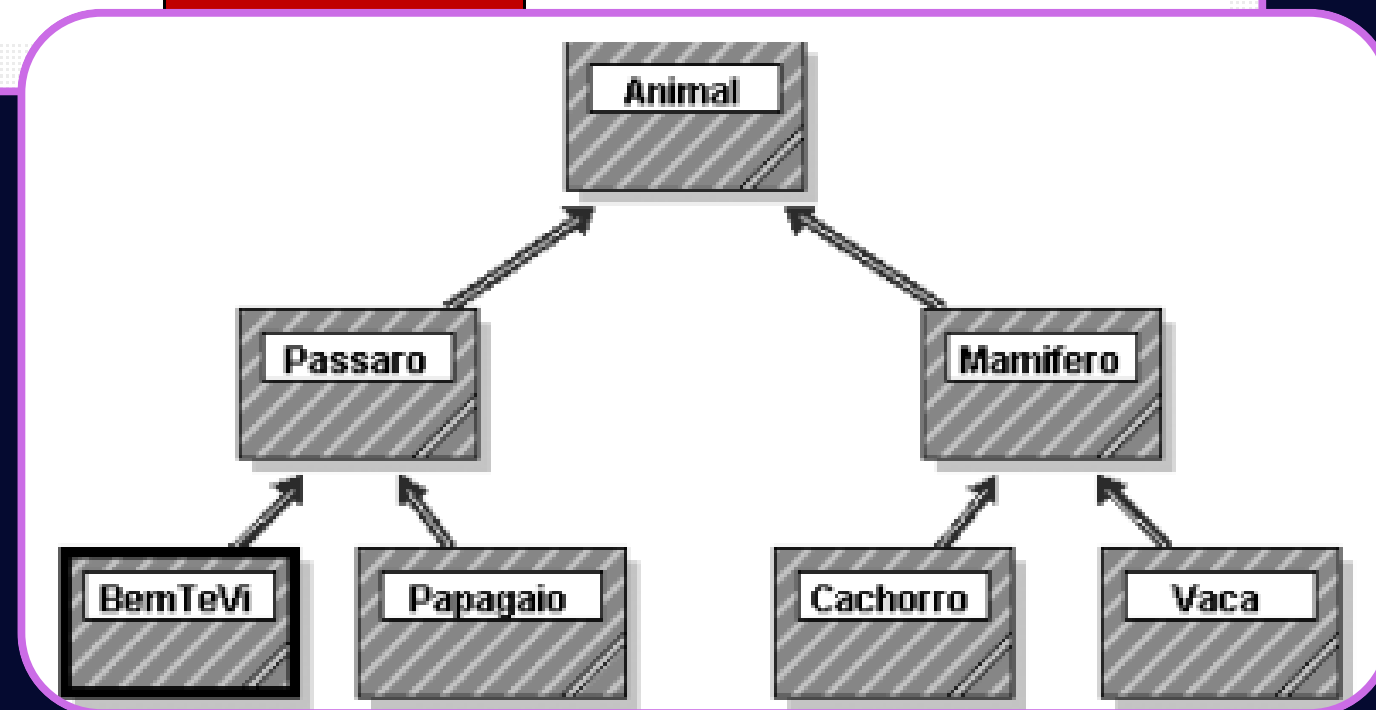
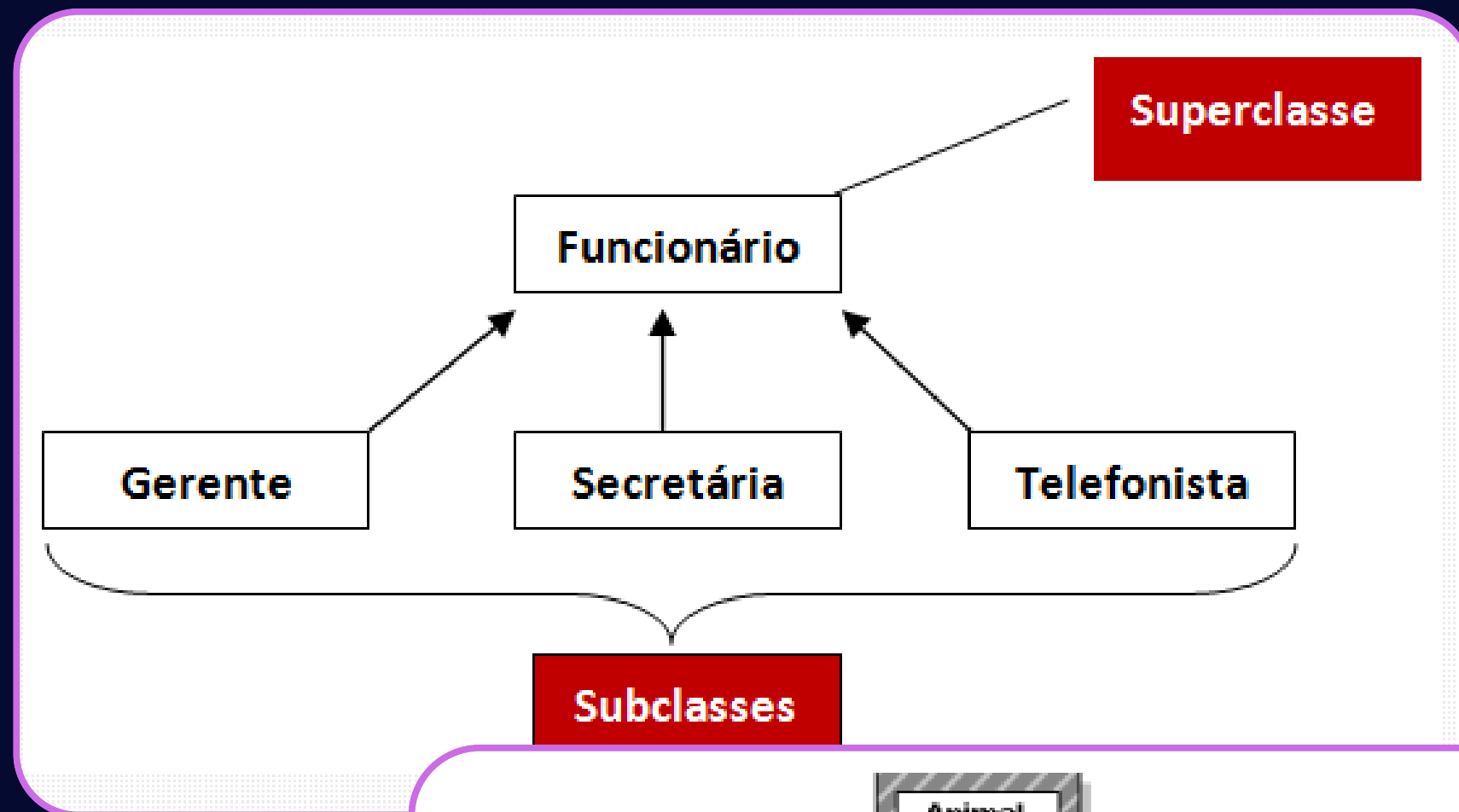
```
public class Main {  
    public static void main(String[] args) {  
        Cachorro cachorro = new Cachorro("Rex");  
        Gato gato = new Gato("Whiskers");  
  
        cachorro.fazerSom(); // Saída: O cachorro late  
        gato.fazerSom();    // Saída: O gato mia  
        System.out.println(cachorro.getNome()); // Saída: Rex  
        System.out.println(gato.getNome());    // Saída: Whiskers  
    }  
}
```

Ninguém adota um animal

```
public abstract class Animal {  
    private String nome;  
  
    public Animal(String nome) {  
        this.nome = nome;  
    }  
  
    public abstract void fazerSom() ;  
  
    public String getNome() {  
        return nome;  
    }  
}
```

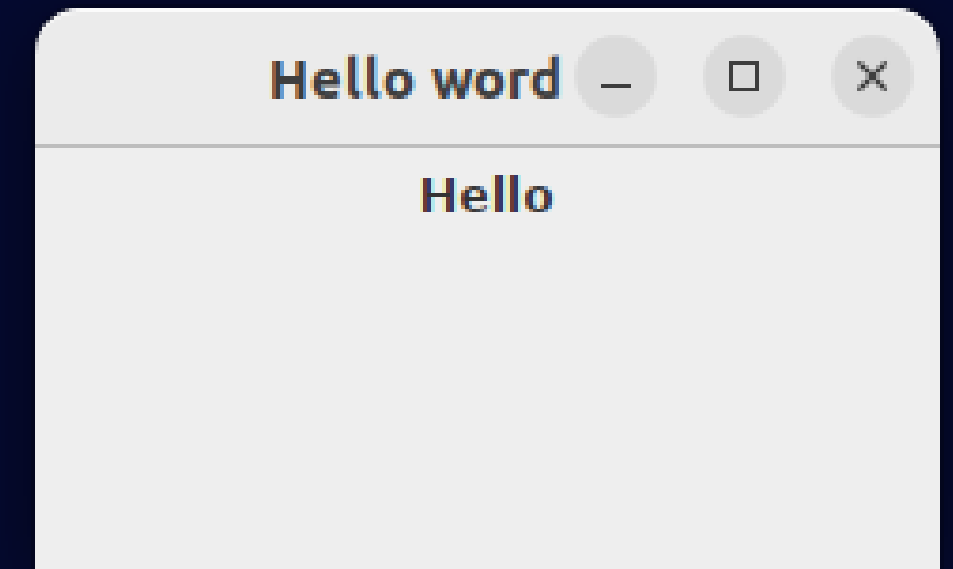


Exemplos clássicos



Outros

```
public class HelloWorld extends JFrame {  
    public HelloWorld() {  
        super("Hello word");  
        JLabel label = new JLabel("Hello");  
        this.setLayout(new FlowLayout());  
        this.add(label);  
        this.setSize(240, 150);  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
  
    public static void main(String[] args) {  
        HelloWorld hello = new HelloWorld();  
        hello.setVisible(true);  
    }  
}
```



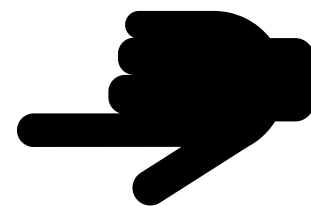
Python

```
class Animal:
    def __init__(self, nome):
        self.nome = nome

    def fazer_som(self):
        print("O animal faz um som indefinido")

    def get_nome(self):
        return self.nome

class Cachorro(Animal):
    def fazer_som(self):
        print("O cachorro late")
```



Cade meu construtor?

Python

```
class Animal:
    def __init__(self, nome):
        self.nome = nome

    def fazer_som(self):
        print("O animal faz um som indefinido")

    def get_nome(self):
        return self.nome

class Cachorro(Animal):
    def fazer_som(self):
        print("O cachorro latou")
```

```
class Gato(Animal):
    def __init__(self, nome, sobre):
        super().__init__(nome)
        self.sobrenome = sobre

    def fazer_som(self):
        print("O gato mia")

    def get_nome(self):
        return self.nome + ' ' + self.sobrenome
```


Python

```
class Animal:
    def __init__(self, nome):
        self.nome = nome

cachorro = Cachorro("Rex")
gato = Gato("Whiskers", "da Silva")

cachorro.fazer_som() # Saída: 0 cachorro late
gato.fazer_som() # Saída: 0 gato mia
print(cachorro.get_nome()) # Saída: Rex
print(gato.get_nome()) # Saída: Wiskers da Silva

print("0 cachorro late")
print("0 gato mia")

def get_nome(self):
    return self.nome + ' ' + self.sobrenome
```

Python abstrato

```
from abc import ABC, abstractmethod
```

```
class Animal(ABC):  
    def __init__(self, nome):  
        self.nome = nome
```

```
@abstractmethod
```

```
def fazer_som(self):  
    pass
```

```
def get_nome(self):  
    return self.nome
```

```
lva")
```

```
0 cachorro late
```

```
0 gato mia
```

```
aída: Rex
```

```
: Wiskers da Silva
```

```
gato mia")
```

```
def get_nome(self):
```

```
    return self.nome + ' ' + self.sobrenome
```

Herança múltipla

```
from abc import ABC, abstractmethod
```

```
class Animal(ABC):  
    def __init__(self,  
        self.nome = nome)  
  
    @abstractmethod  
    def fazer_som(self)  
        pass  
  
    def get_nome(self):  
        return self.nome
```

```
public class HelloWorld extends JFrame {  
  
    public HelloWorld() {  
        super("Hello word");  
        JLabel label = new JLabel("Hello");  
        this.setLayout(new FlowLayout());  
        this.add(label);  
        this.setSize(240, 150);  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
  
    public static void main(String[] args) {  
        HelloWorld hello = new HelloWorld();  
        hello.setVisible(true);  
    }  
}
```

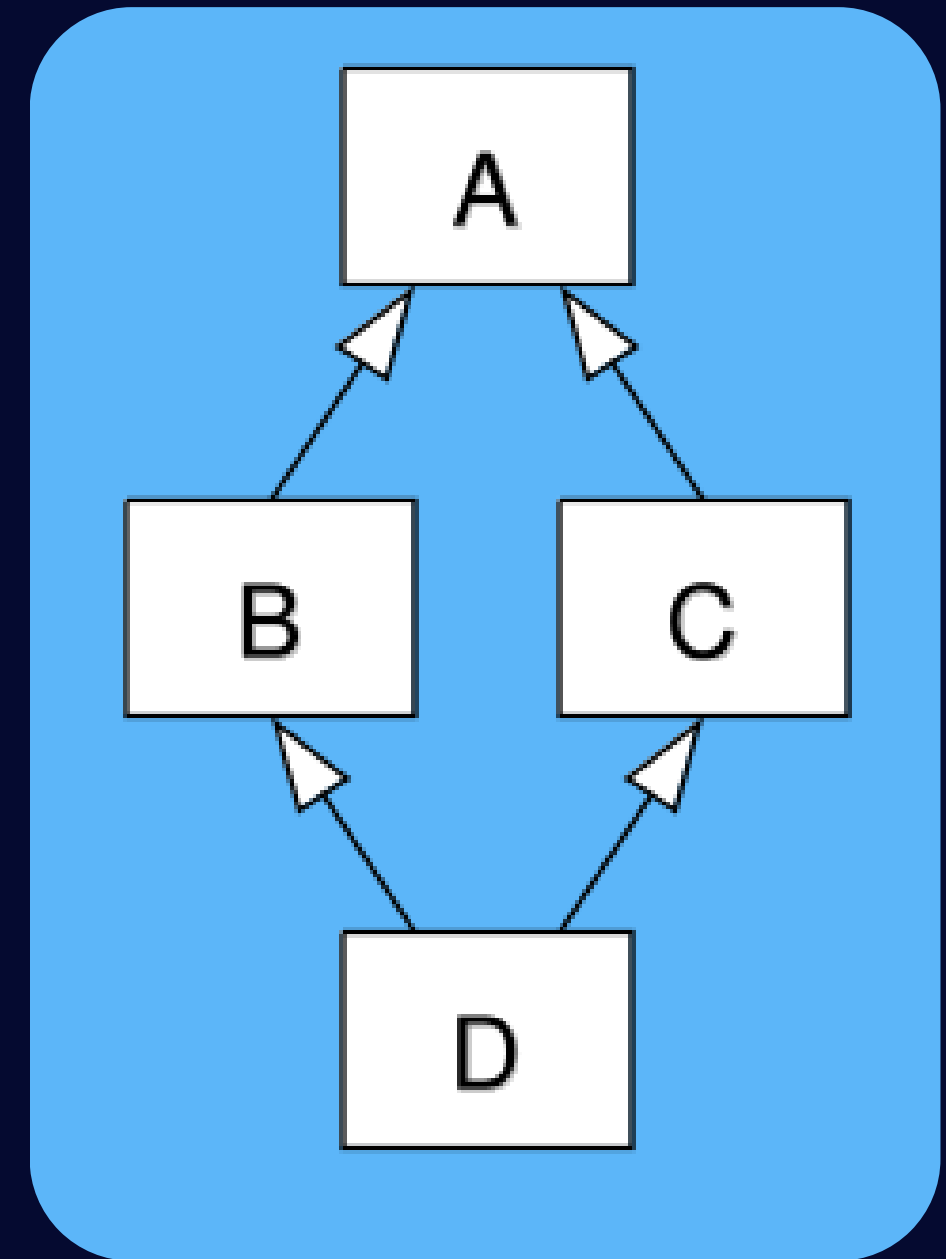
Herança múltipla

```
class A:
```

```
class B(A):
```

```
class C(A):
```

```
class D(B, C):
```



Herança múltipla

```
class A:
```

```
class B(A):
```

```
class C(A):
```

```
class D(B, C):
```

```
class A:  
    def m1(self):  
        print('A.m1')
```

```
class B(A):  
    def m1(self):  
        print('B.m1')
```

```
class C(A):  
    def m1(self):  
        print('C.m1')
```

```
class D(B,C):  
    pass
```

```
d = D()  
d.m1()
```

Herança múltipla

```
class A:
```

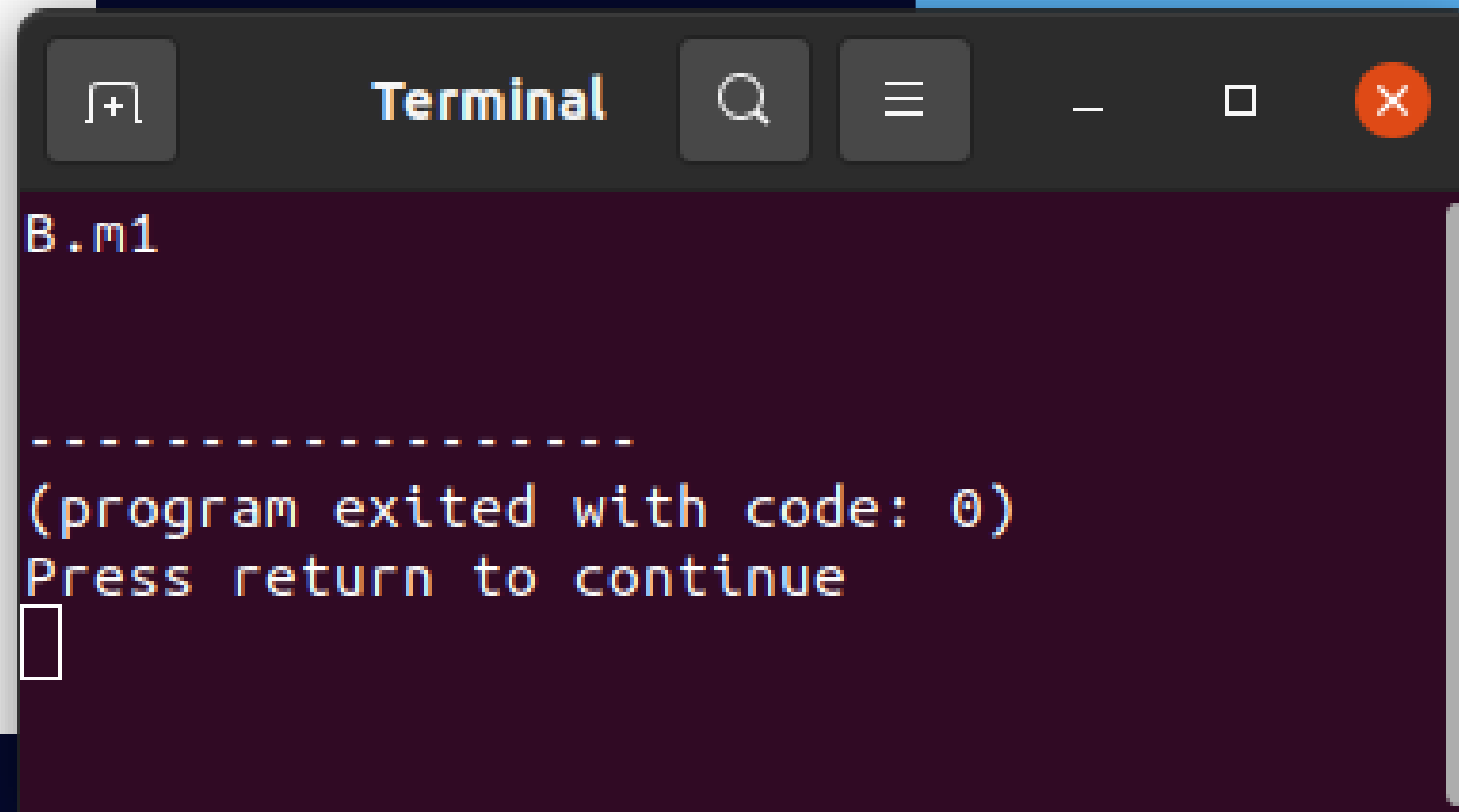
```
class B(A):
```

```
class C(A):
```

```
class D(B, C):
```

```
class A:  
    def m1(self):  
        print('A.m1')
```

```
class B(A):  
    def m1(self):  
        print('B.m1')
```



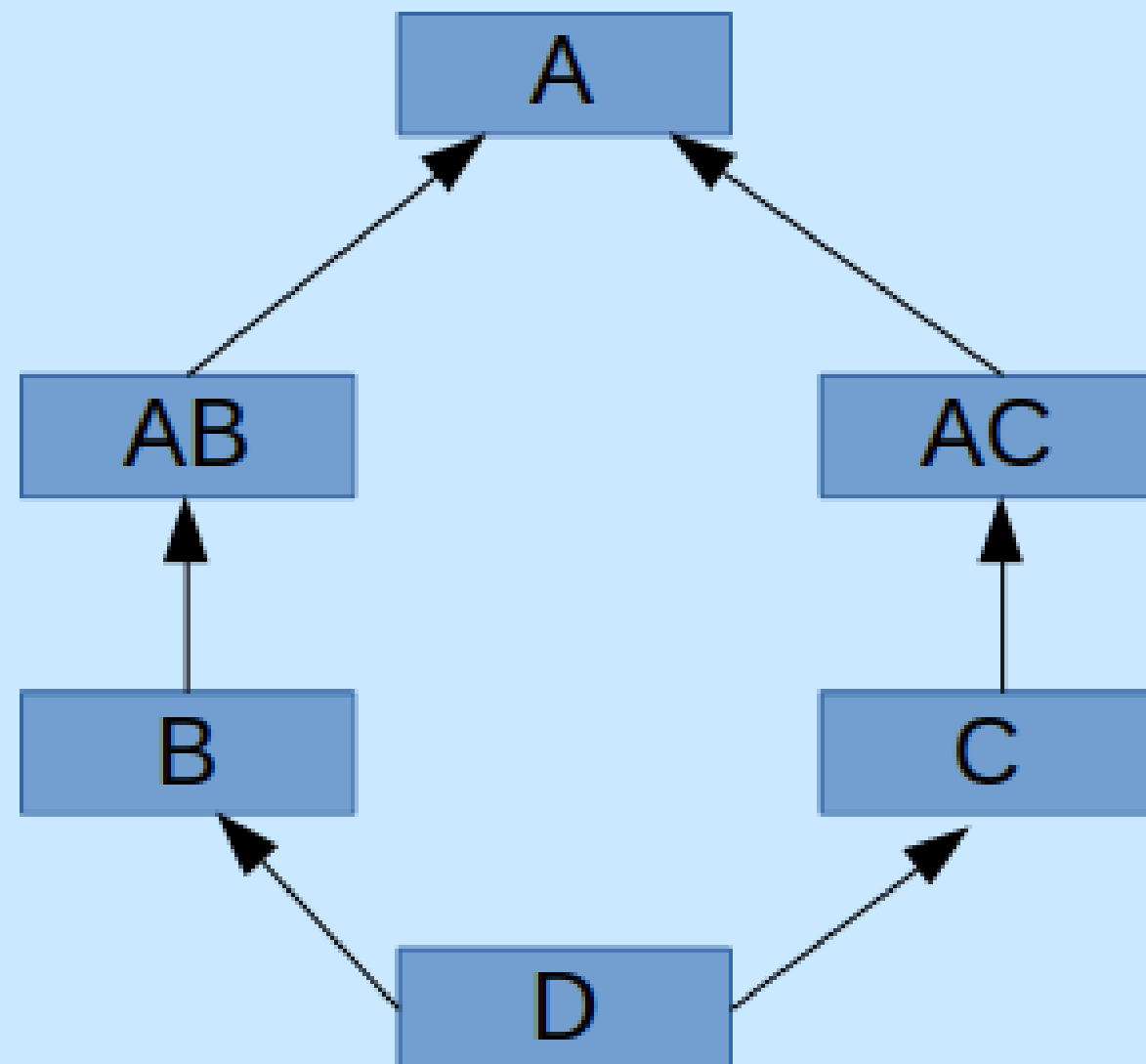
```
B.m1  
  
-----  
(program exited with code: 0)  
Press return to continue  
█
```

MRO – Ordem de resolução de métodos

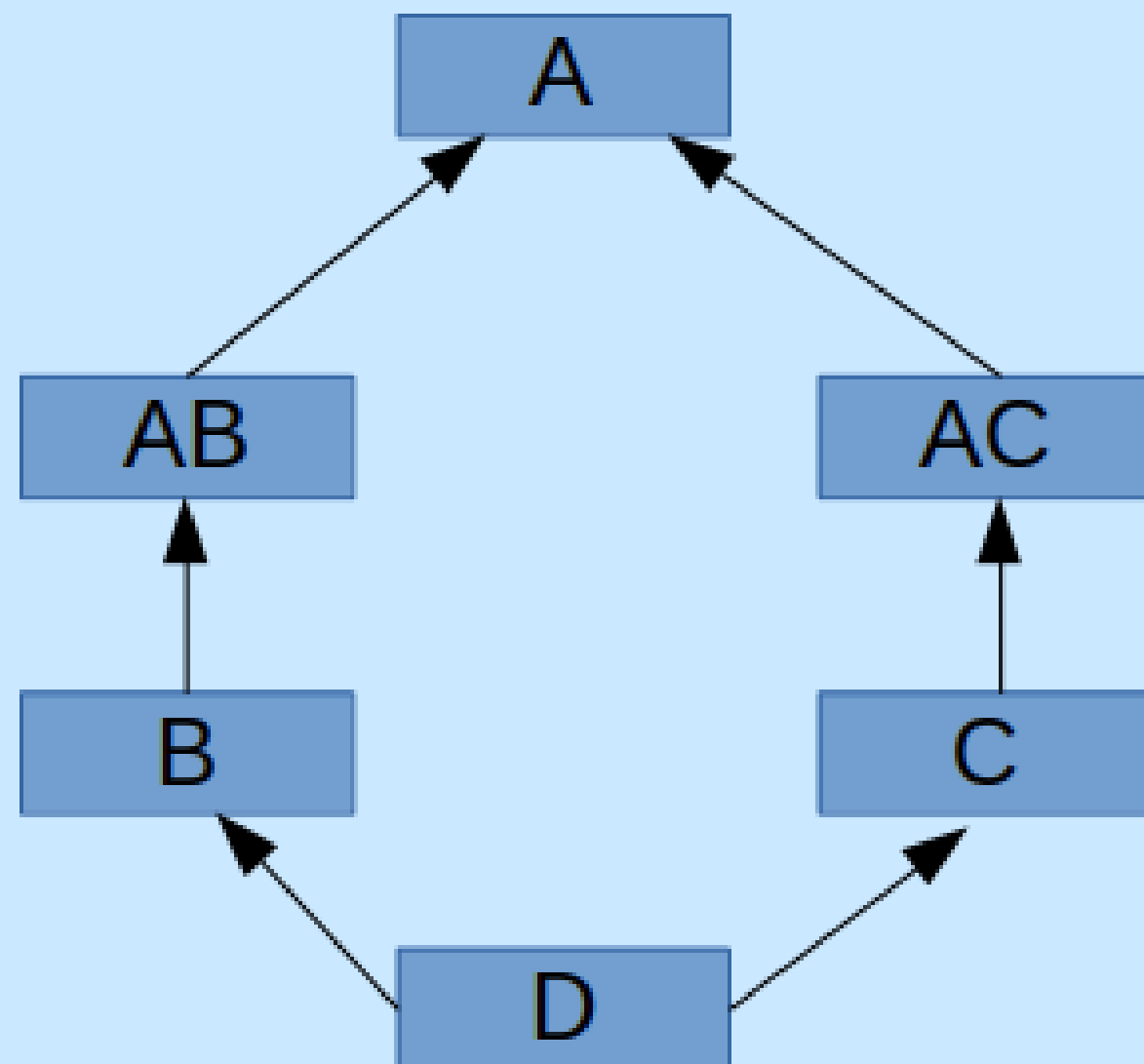
- Estabelece qual a ordem de 'procura' do método na hierarquia
- Da esquerda para a direita
- Para conhecer a ordem de uma classe pode-se usar o método `mro()`.

```
D.mro() ==> [<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class '__main__.A'>, <class 'object'>]
```


MRO – Ordem de resolução de métodos

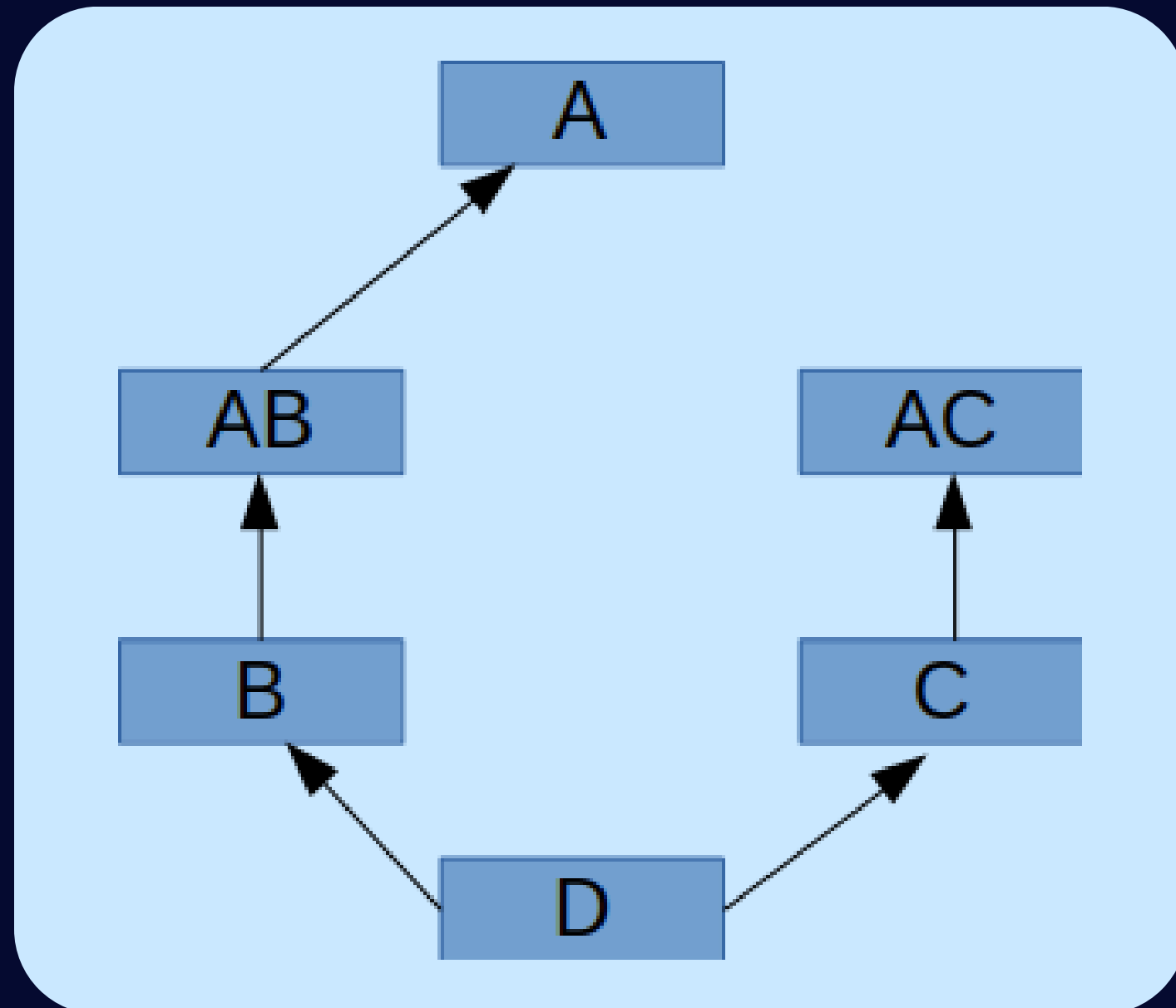


MRO – Ordem de resolução de métodos

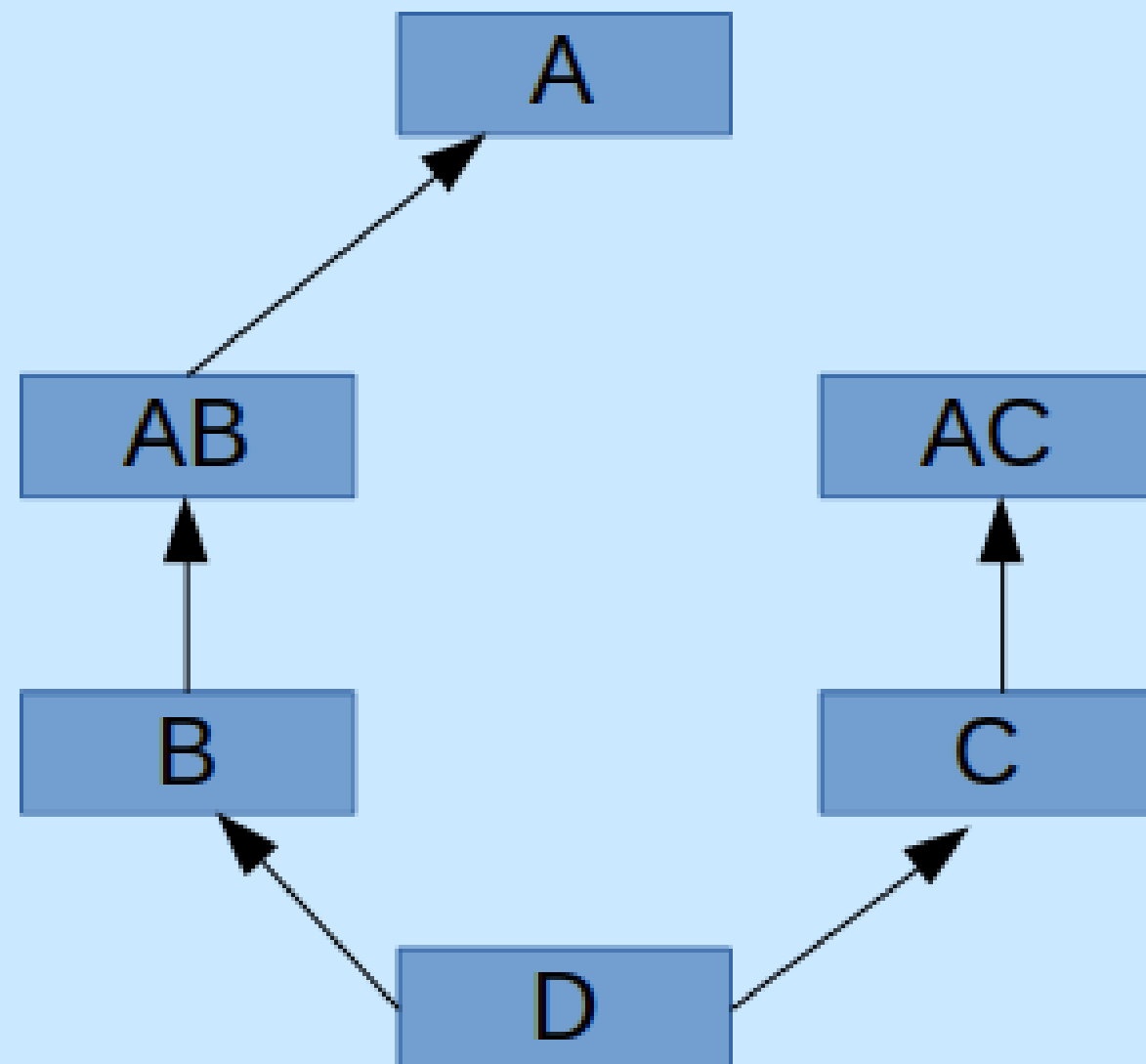


```
[<class '__main__.D'>,  
<class '__main__.B'>,  
<class '__main__.AB'>,  
<class '__main__.C'>,  
<class '__main__.AC'>,  
<class '__main__.A'>,  
<class 'object'>]
```

MRO – Ordem de resolução de métodos

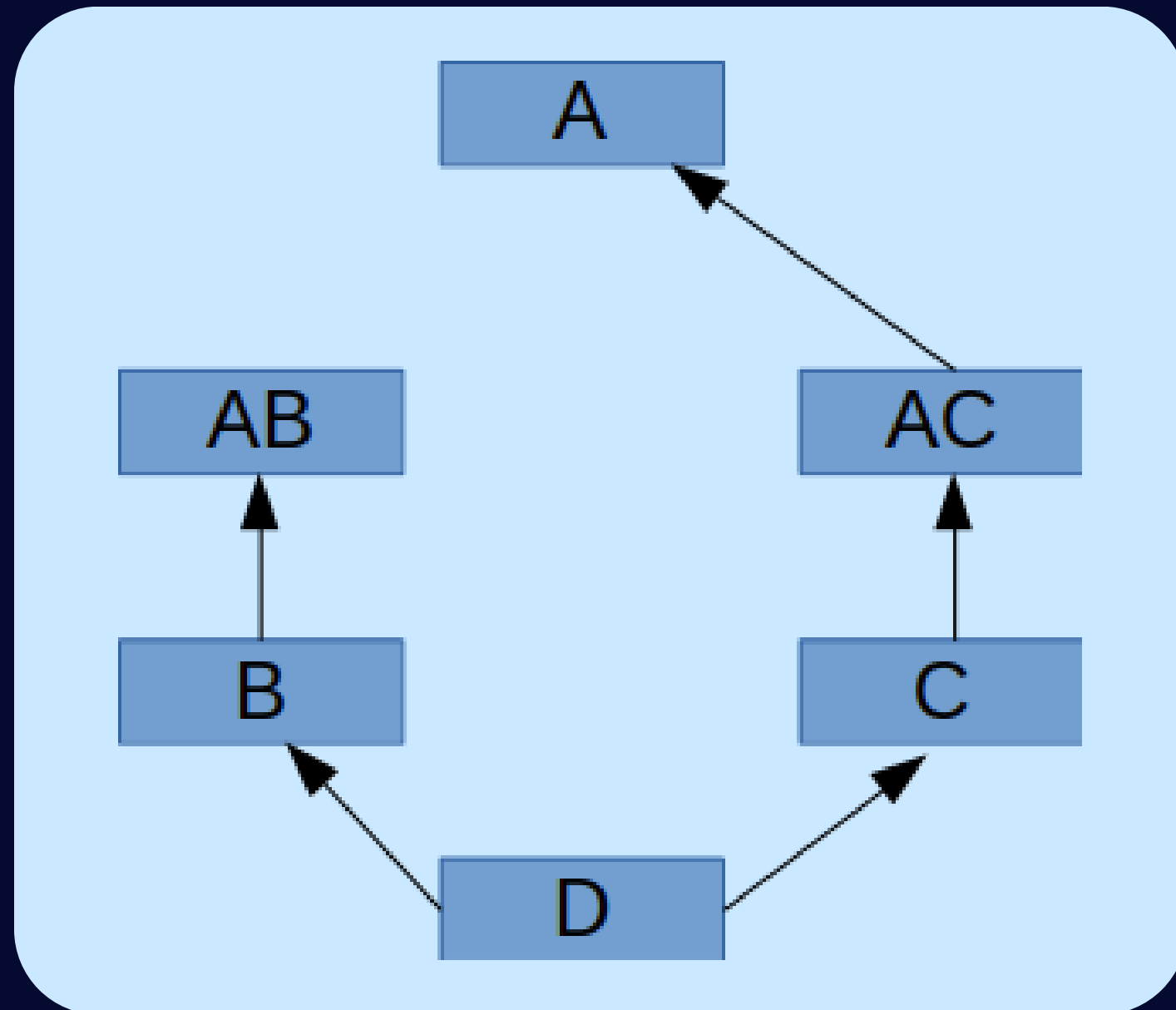


MRO – Ordem de resolução de métodos

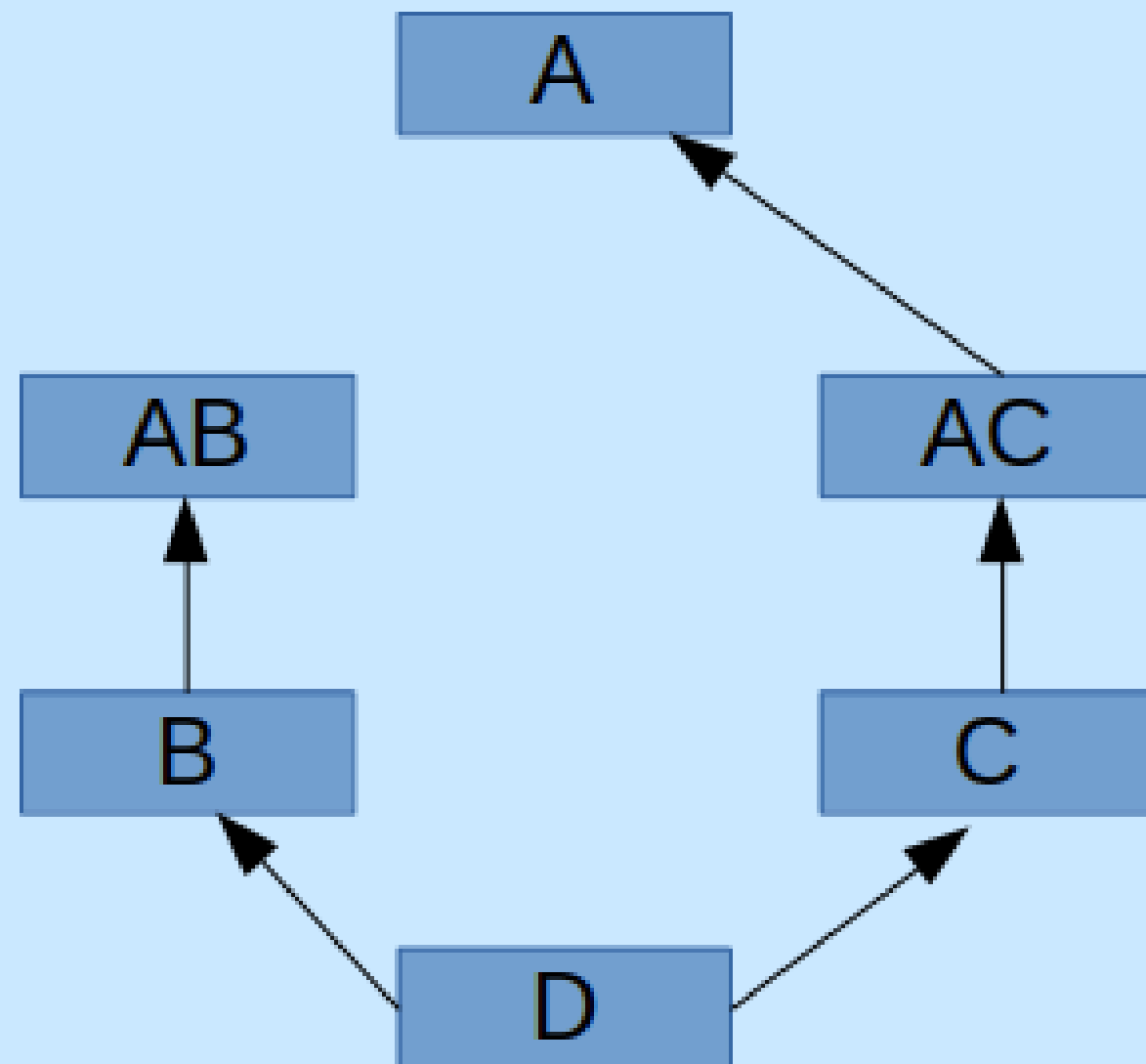


```
[<class '__main__.D'>,
 <class '__main__.B'>,
 <class '__main__.AB'>,
 <class '__main__.A'>,
 <class '__main__.C'>,
 <class '__main__.AC'>,
 <class 'object'>]
```

MRO – Ordem de resolução de métodos



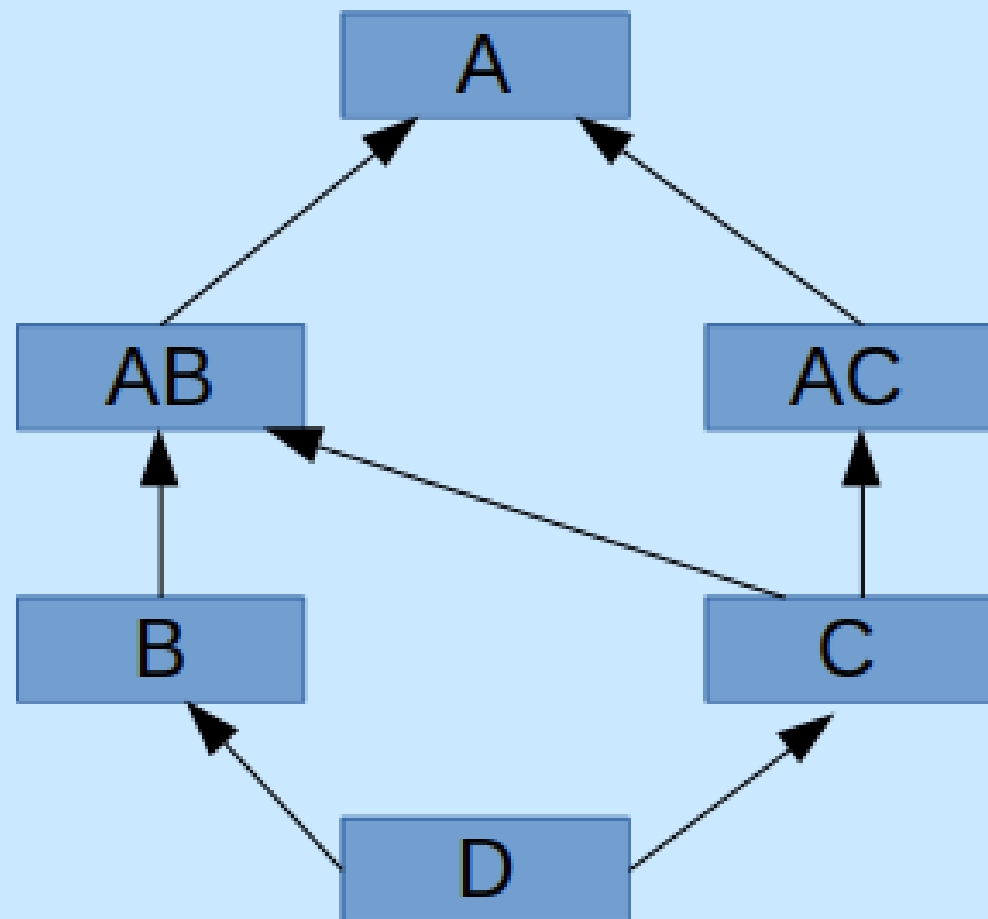
MRO – Ordem de resolução de métodos



```
[<class '__main__.D'>,
 <class '__main__.B'>,
 <class '__main__.AB'>,
 <class '__main__.C'>,
 <class '__main__.AC'>,
 <class '__main__.A'>,
 <class 'object'>]
```

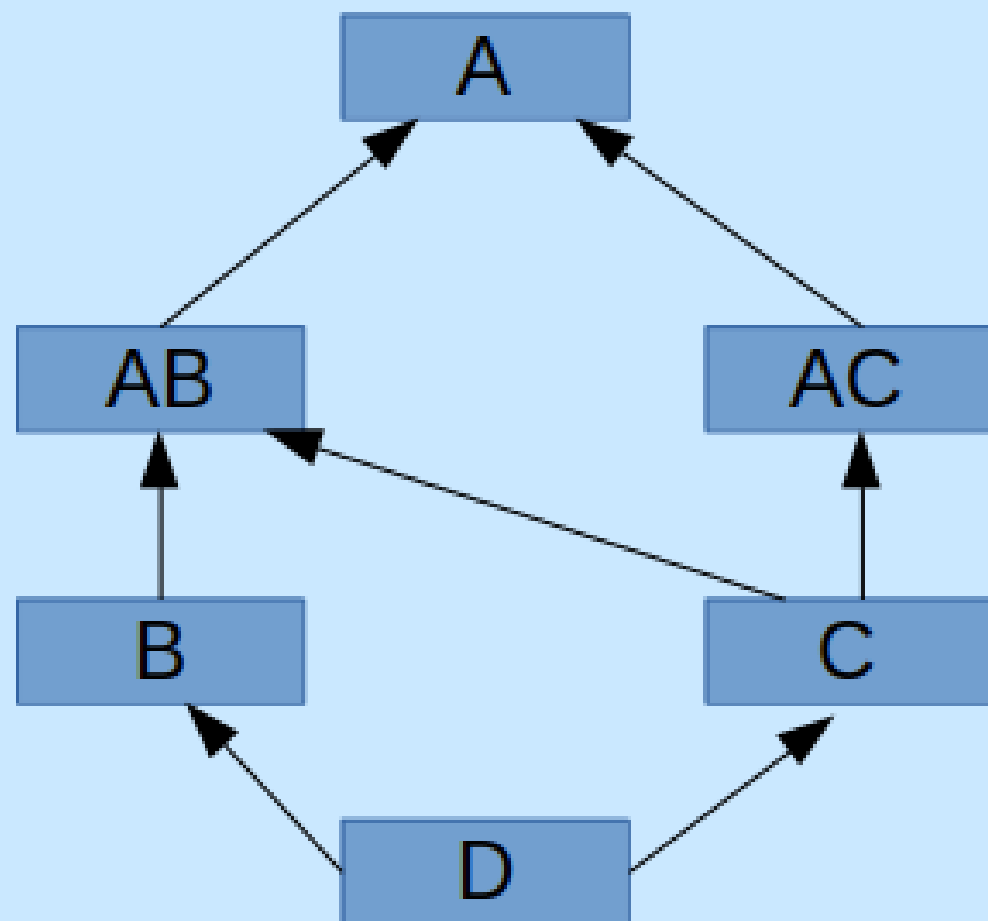
MRO – Ordem de resolução de métodos

```
class C(AC,AB):
```



MRO – Ordem de resolução de métodos

class C(AC,AB):



```
[<class '__main__.D'>,  
<class '__main__.B'>,  
<class '__main__.C'>,  
<class '__main__.AC'>,  
<class '__main__.AB'>,  
<class '__main__.A'>,  
<class 'object'>]
```

Problema dos construtores

- Esse problema vale para construtores
- Para complicar, construtores de classes diferentes podem ter parâmetros diferentes
- Nesse caso, não tem como passar os parâmetros corretos para os construtores corretos